Boosting Textual Compression
(2005; Ferragina, Giancarlo, Manzini, Sciortino)

Paolo Ferragina, University of Pisa, www.di.unipi.it/~ferragin
Giovanni Manzini, University of Piemonte Orientale, www.mfn.unipmn.it/~manzini

Entry Editor: Paolo Ferragina

INDEX TERMS: Data compression, Empirical Entropy, Burrows-Wheeler Transform, Huffman and Arithmetic Coding.

SYNONYMS:.

1 PROBLEM DEFINITION

Informally, a boosting technique is a method that, when applied to a particular class of algorithms, yields improved algorithms. The improvement must be provable and well defined in terms of one or more of the parameters characterizing the algorithmic performance. Examples of boosters can be found in the context of Randomized Algorithms (here, a booster allows to turn a BPP algorithm into an RP one [5]) and Computational Learning Theory (here, a booster allows to improve the prediction accuracy of a weak learning algorithm [10]). The problem of Compression Boosting consists in designing a technique that improves the compression performance of a wide class of algorithms. In particular, the result of Ferragina *et al.* consists in a general technique for turning a compressor that uses no context information into one that always uses the best possible context.

The classic Huffman and Arithmetic coding algorithms [1] are examples of *statistical* compressors which typically encode an input symbol according to its *overall* frequency in the data to be compressed.[1] This approach is efficient and easy to implement but achieves poor compression. The compression performance of statistical compressors can be improved by adopting *higher*-order models that obtain better estimates for the frequencies of the input symbols. The PPM compressor [9] implements this idea by collecting (the frequency of) all symbols which follow *any* k-long context, and by compressing them via Arithmetic coding. The length k of the context is a parameter of the algorithm that depends on the data to be compressed: it is different if one is compressing English text, a DNA sequence, or an XML document. There exist other examples of sophisticated compressors that use context information in an *implicit* way, such as Lempel-Ziv and Burrows-Wheeler compressors [9]. All these context-aware algorithms are effective in terms of compression performance, but are usually rather complex to implement and difficult to analyze.

Applying the boosting technique of Ferragina *et al.* to Huffman or Arithmetic Coding yields a new compression algorithm with the following features: (*i*) the new algorithm uses the boosted compressor as a black box, (*ii*) the new algorithm compresses in a PPM-like style, automatically choosing the *optimal* value of k, (*iii*) the new algorithm has essentially the same time/space asymptotic performance of the boosted compressor. The following sections give a precise and formal treatment of the three properties (*i*)–(*iii*) outlined above.

[1] In their dynamic versions these algorithms consider the frequency of a symbol in the already scanned portion of the input.

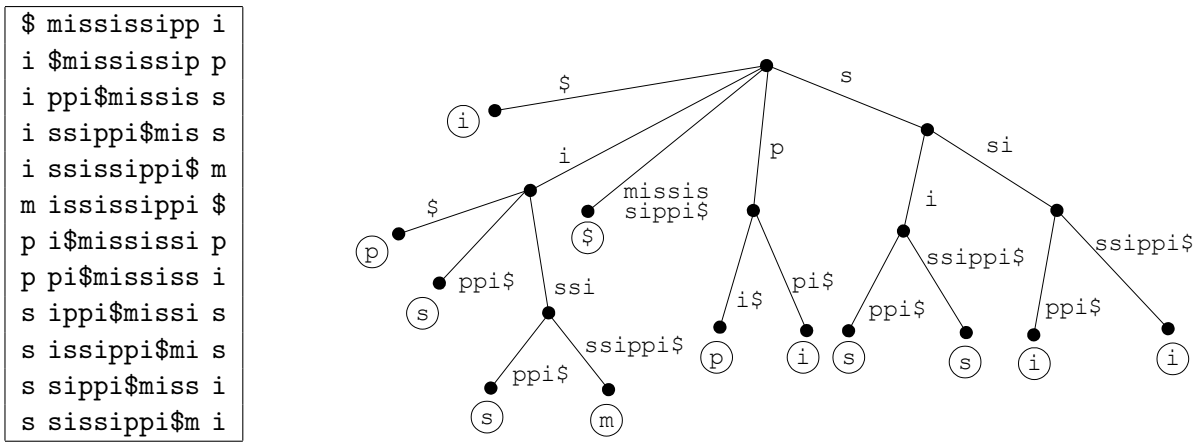

Figure 1: The bwt matrix (left) and the suffix tree (right) for the string $s = \texttt{mississippi\$}$. The output of the bwt is the last column of the bwt matrix, i.e., $\hat{s} = \mathsf{bwt}(s) = \texttt{ipssm\$pissii}$.

2 KEY RESULTS

Notation: The empirical entropy. Let s be a string over the alphabet $\Sigma = \{a_1, \ldots, a_h\}$ and, for each $a_i \in \Sigma$, let n_i be the number of occurrences of a_i in s. The 0-*th order empirical entropy* of the string s is defined as $H_0(s) = -\sum_{i=1}^{h}(n_i/|s|)\log(n_i/|s|)$, where it is assumed that all logarithms are taken to the base 2 and $0 \log 0 = 0$. It is well known that H_0 is the maximum compression one can achieve using a uniquely decodable code in which a fixed codeword is assigned to each alphabet symbol. Greater compression is achievable if the codeword of a symbol depends on the k symbols following it (namely, its *context*).[2] Let us define w_s as the string of single symbols immediately preceding the occurrences of w in s. For example, for $s = \texttt{bcabcabdca}$ it is $\texttt{ca}_s = \texttt{bbd}$. The value

$$H_k(s) = \frac{1}{|s|} \sum_{w \in \Sigma^k} |w_s| \, H_0(w_s) \tag{1}$$

is the k-th order empirical entropy of s and is a lower bound to the compression one can achieve using codewords which only depend on the k symbols immediately following the one to be encoded.

Example 1 Let $s = \texttt{mississippi}$. For $k = 1$ it is $\texttt{i}_s = \texttt{mssp}$, $\texttt{s}_s = \texttt{isis}$, $\texttt{p}_s = \texttt{ip}$. Hence, $H_1(s) = \frac{4}{11}H_0(\texttt{mssp}) + \frac{4}{11}H_0(\texttt{isis}) + \frac{2}{11}H_0(\texttt{ip}) = \frac{6}{11} + \frac{4}{11} + \frac{2}{11} = \frac{12}{11}$. □

Note that the empirical entropy is defined for any string and can be used to measure the performance of compression algorithms without any assumption on the input source. Unfortunately, for some (highly compressible) strings, the empirical entropy provides a lower bound that is too conservative. For example, for $s = \texttt{a}^n$ it is $|s| \, H_k(s) = 0$ for any $k \geq 0$. To better deal with highly compressible strings [6] introduced the notion of 0-*th order modified empirical* entropy $H_0^*(s)$ whose property is that $|s|H_0^*(s)$ is at least equal to the number of bits needed to write down the length of s in binary. The *k-th order modified empirical entropy* H_k^* is then defined in terms of H_0^* as the maximum compression one can achieve by looking at *no more than* k symbols following the one to be encoded.

[2] In data compression it is customary to define the context looking at the symbols *preceding* the one to be encoded. The present entry uses the non-standard "forward" contexts to simplify the notation of the following sections. Note that working with "forward" contexts is equivalent to working with the traditional "backward" contexts on the string s reversed (see [4] for details).

The Burrows-Wheeler Transform. Given a string s, the Burrows-Wheeler transform [2] (bwt for short) consists of three basic steps: (1) append to the end of s a special symbol \$ smaller than any other symbol in Σ; (2) form a *conceptual* matrix $\mathcal{M}$ whose rows are the cyclic shifts of the string $s\$$, sorted in lexicographic order; (3) construct the transformed text $\hat{s} = \texttt{bwt}(s)$ by taking the last column of $\mathcal{M}$ (see Figure 1). In [2] Burrows and Wheeler proved that $\hat{s}$ is a permutation of s, and that from $\hat{s}$ it is possible to recover s in $O(|s|)$ time.

To see the power of the bwt the reader should reason in terms of empirical entropy. Fix a positive integer k. The first k columns of the bwt matrix contain, lexicographically ordered, all length-k substrings of s (and k substrings containing the symbol \$). For any length-k substring w of s, the symbols immediately preceding every occurrence of w in s are grouped together in a set of consecutive positions of $\hat{s}$ since they are the last symbols of the rows of $\mathcal{M}$ prefixed by w. Using the notation introduced for defining H_k, it is possible to rephrase this property by saying that the symbols of w_s are consecutive within $\hat{s}$, or equivalently, that $\hat{s}$ contains, as a substring, a permutation $\pi_w(w_s)$ of the string w_s.

Example 2 Let $s = \texttt{mississippi}$ and $k = 1$. Figure 1 shows that $\hat{s}[1,4] = \texttt{pssm}$ is a permutation of $\texttt{i}_s = \texttt{mssp}$. In addition, $\hat{s}[6,7] = \texttt{pi}$ is a permutation of $\texttt{p}_s = \texttt{ip}$, and $\hat{s}[8,11] = \texttt{ssii}$ is a permutation of $\texttt{s}_s = \texttt{isis}$. □

Since permuting a string does not change its (modified) zeroth order empirical entropy (that is, $H_0(\pi_w(w_s)) = H_0(w_s)$), the Burrows-Wheeler transform can be seen as a tool for reducing the problem of compressing s up to its k-th order entropy to the problem of compressing *distinct portions* of $\hat{s}$ up to their *zero-th order* entropy. To see this, assume to partition $\hat{s}$ into the substrings $\pi_w(w_s)$ by varying w over Σ^k. It follows that $\hat{s} = \bigsqcup_{w\in\Sigma^k} \pi_w(w_s)$ where $\bigsqcup$ denotes the concatenation operator among strings.[3] By (1) it follows that

$$\sum_{w\in\Sigma^k} |\pi_w(w_s)|\, H_0(\pi_w(w_s)) = \sum_{w\in\Sigma^k} |w_s|\, H_0(w_s) = |s|\, H_k(s).$$

Hence, to compress s up to $|s|\, H_k(s)$ it suffices to compress each substring $\pi_w(w_s)$ up to its zeroth order empirical entropy. Note however that in the above scheme the parameter k must be chosen in advance. Moreover, a similar scheme cannot be applied to H_k^* which is defined in terms of contexts of length *at most* k. As a result, no efficient procedure is known for computing the partition of $\hat{s}$ corresponding to $H_k^*(s)$. The compression booster [4] is a natural complement to the bwt and allows to compress any string s up to $H_k(s)$ (or $H_k^*(s)$) simultaneously for all $k \geq 0$.

The Compression Boosting Algorithm. A crucial ingredient of compression boosting is the relationship between the bwt matrix and the suffix tree data structure. Let $\mathcal{T}$ denote the suffix tree of the string $s\$$. $\mathcal{T}$ has $|s|+1$ leaves, one per suffix of $s\$$, and edges labeled with substrings of $s\$$ (see Figure 1). Any node u of $\mathcal{T}$ has *implicitly associated* a substring of $s\$$, given by the concatenation of the edge labels on the downward path from the root of $\mathcal{T}$ to u. In that implicit association, the leaves of $\mathcal{T}$ correspond to the suffixes of $s\$$. Assume that the suffix tree edges are sorted lexicographically. Since each row of the bwt matrix is prefixed by one suffix of $s\$$ and rows are lexicographically sorted, the i-th leaf (counting from the left) of the suffix tree corresponds to the i-th row of the bwt matrix. Associate to the i-th leaf of $\mathcal{T}$ the i-th symbol of $\hat{s} = \texttt{bwt}(s)$. In Figure 1 these symbols are represented inside circles.

For any suffix tree node u, let $\hat{s}\langle u\rangle$ denote the substring of $\hat{s}$ obtained by concatenating, from left to right, the symbols associated to the leaves descending from node u. Of course $\hat{s}\langle root(\mathcal{T})\rangle = \hat{s}$. A subset $\mathcal{L}$ of $\mathcal{T}$'s nodes is called a *leaf cover* if every leaf of the suffix tree has a *unique* ancestor in

[3] In addition to $\sqcup_{w\in\Sigma^k}\pi_w(w_s)$, the string $\hat{s}$ also contains the last k symbols of s (which do not belong to any w_s) and the special symbol \$. For simplicity these symbols will be ignored in the following part of the entry.

$\mathcal{L}$. Any leaf cover $\mathcal{L} = \{u_1, \ldots, u_p\}$ naturally induces a partition of the leaves of $\mathcal{T}$. Because of the relationship between $\mathcal{T}$ and the bwt matrix this is also a partition of $\hat{s}$, namely $\{\hat{s}\langle u_1\rangle, \ldots, \hat{s}\langle u_p\rangle\}$.

Example 3 Consider the suffix tree in Figure 1. A leaf cover consists of all nodes of depth one. The partition of $\hat{s}$ induced by this leaf cover is $\{\texttt{i}, \texttt{pssm}, \$, \texttt{pi}, \texttt{ssii}\}$. □

Let C denote a function that associates to every string x over $\Sigma \cup \{\$\}$ a positive real value $C(x)$. For any leaf cover $\mathcal{L}$, define its cost as $C(\mathcal{L}) = \sum_{u \in \mathcal{L}} C(\hat{s}\langle u\rangle)$. In other words, the cost of the leaf cover $\mathcal{L}$ is equal to the sum of the costs of the strings in the partition induced by $\mathcal{L}$. A leaf cover $\mathcal{L}_{\min}$ is called *optimal* with respect to C if $C(\mathcal{L}_{\min}) \leq C(\mathcal{L})$, for any leaf cover $\mathcal{L}$.

Let A be a compressor such that, for any string x, its output size is bounded by $|x|H_0(x)+\eta|x|+\mu$ bits, where η and μ are constants. Define the cost function $C_{\texttt{A}}(x) = |x|H_0(x) + \eta|x| + \mu$. In [4] Ferragina *et al.* exhibit a linear-time greedy algorithm that computes the optimal leaf cover $\mathcal{L}_{\min}$ with respect to $C_{\texttt{A}}$. The authors of [4] also show that, for any $k \geq 0$, there exists a leaf cover $\mathcal{L}_k$ of cost $C_{\texttt{A}}(\mathcal{L}_k) = |s|\, H_k(s) + \eta|s| + O(|\Sigma|^k)$. These two crucial observations show that, if one uses A to compress each substring in the partition induced by the optimal leaf cover $\mathcal{L}_{\min}$, the total output size is bounded in terms of $|s|\, H_k(s)$, for any $k \geq 0$. In fact,

$$\sum_{u \in \mathcal{L}_{\min}} C_{\texttt{A}}(\hat{s}\langle u\rangle) = C_{\texttt{A}}(\mathcal{L}_{\min}) \leq C_{\texttt{A}}(\mathcal{L}_k) = |s|\, H_k(s) + \eta|s| + O(|\Sigma|^k)$$

In summary, boosting the compressor A over the string s consists of three main steps:

1. Compute $\hat{s} = \mathsf{bwt}(s)$;
2. Compute the optimal leaf cover $\mathcal{L}_{\min}$ with respect to $C_{\texttt{A}}$, and partition $\hat{s}$ according to $\mathcal{L}_{\min}$;
3. Compress each substring of the partition using the algorithm A.

So the boosting paradigm reduces the design of effective compressors that use context information, to the (usually easier) design of zeroth order compressors. The performance of this paradigm is summarized by the following theorem.

Theorem 1. [Ferragina et al. 2005] *Let* A *be a compressor that squeezes any string x in at most $|x|\, H_0(x) + \eta|x| + \mu$ bits. The compression booster applied to* A *produces an output whose size is bounded by $|s|\, H_k(s) + \log|s| + \eta|s| + O(|\Sigma|^k)$ bits simultaneously for all $k \geq 0$. With respect to* A*, the booster introduces a space overhead of $O(|s| \log|s|)$ bits and no asymptotic time overhead in the compression process.* □

A similar result holds for the modified entropy H_k^* as well (but it is much harder to prove): Given a compressor A that squeezes any string x in at most $\lambda|x|\, H_0^*(x) + \mu$ bits, the compression booster produces an output whose size is bounded by $\lambda|s|H_k^*(s)+\log|s|+O(|\Sigma|^k)$ bits, simultaneously for all $k \geq 0$. In [4] the authors also show that no compression algorithm, satisfying some mild assumptions on its inner working, can achieve a similar bound in which both the multiplicative factor λ and the additive logarithmic term are dropped simultaneously. Furthermore [4] proposes an instantiation of the booster which compresses any string s in at most $2.5|s|H_k^*(s) + \log|s| + O(|\Sigma|^k)$ bits. This bound is analytically superior to the bounds proven for the best existing compressors including Lempel-Ziv, Burrows-Wheeler, and PPM compressors.

3 APPLICATIONS

Apart from the natural application in data compression, compressor boosting has been used also to design Compressed Full-text Indexes [7].

4 OPEN PROBLEMS

The boosting paradigm may be generalized as follows: Given a compressor A, find a permutation $\mathcal{P}$ for the symbols of the string s *and* a partitioning strategy such that the boosting approach, applied to them, minimizes the output size. These pages have provided convincing evidence that the Burrows-Wheeler Transform is an elegant and efficient permutation $\mathcal{P}$. Surprisingly enough, other classic Data Compression problems fall into this framework: Shortest Common Superstring (which is MAX-SNP hard), Run Length Encoding for a Set of Strings (which is polynomially solvable), LZ77 and minimum number of phrases (which is MAX-SNP-Hard). Therefore the boosting approach is general enough to deserve further theoretical and practical attention [8].

5 EXPERIMENTAL RESULTS

An investigation of several compression algorithms based on boosting, and a comparison with other state-of-the-art compressors is presented in [3]. The experiments show that the boosting technique is more robust than other bwt-based approaches, and works well even with less effective zeroth order compressors. However, these positive features are achieved using more (time and space) resources.

6 DATA SETS

The data set used in [3] are available from http://www.mfn.unipmn.it/~manzini/boosting. Other data sets for compression and indexing are available at the Pizza&Chili site http://pizzachili.di.unipi.it/.

7 URL to CODE

The Compression Boosting page (http://www.mfn.unipmn.it/~manzini/boosting) contains the source code of all the algorithms tested in [3]. The code is organized in a highly modular library that can be used to boost any compressor even without knowing the bwt or the boosting procedure.

8 CROSS REFERENCES

Arithmetic coding, Burrows-Wheeler Transform, Compressed text indexing, Table compression, Tree Compression and Indexing.

9 RECOMMENDED READING

[1] T. C. Bell, J. G. Cleary, and I. H. Witten, *Text compression*, Prentice Hall, NJ, 1990.

[2] M. Burrows and D. Wheeler, *A block sorting lossless data compression algorithm*, Tech. Report 124, Digital Equipment Corporation, 1994.

[3] P. Ferragina, R. Giancarlo, and G. Manzini, *The engineering of a compression boosting library: Theory vs practice in bwt compression*, in Proc. 14th European Symposium on Algorithms (ESA), Springer Verlag LNCS n. 4168, 2006, pp. 756–767.

[4] P. Ferragina, R. Giancarlo, G. Manzini, and M. Sciortino, *Boosting textual compression in optimal linear time*, Journal of the ACM, 52 (2005), pp. 688–713.

[5] R. Karp, N. Pippenger, and M. Sipser, *A Time-Randomness tradeoff*, in Proc. Conference on Probabilistic Computational Complexity, AMS, 1985, pp. 150–159.

[6] G. Manzini, *An analysis of the Burrows-Wheeler transform*, Journal of the ACM, 48 (2001), pp. 407–430.

[7] G. Navarro and V. Mäkinen, *Compressed full text indexes*, ACM Computing Surveys, 39(1): 2007.

[8] R. Giancarlo and A. Restivo and M. Sciortino, *From first principles to the Burrows and Wheeler transform and beyond, via combinatorial optimization*, Theoretical Computer Science, 2007 (to appear).

[9] D. Salomon, *Data Compression: the Complete Reference, 3rd Edition*, Springer Verlag, 2004.

[10] R. E. Schapire, *The strength of weak learnability*, Machine Learning, 2 (1990), pp. 197–227.